

FIG. 1

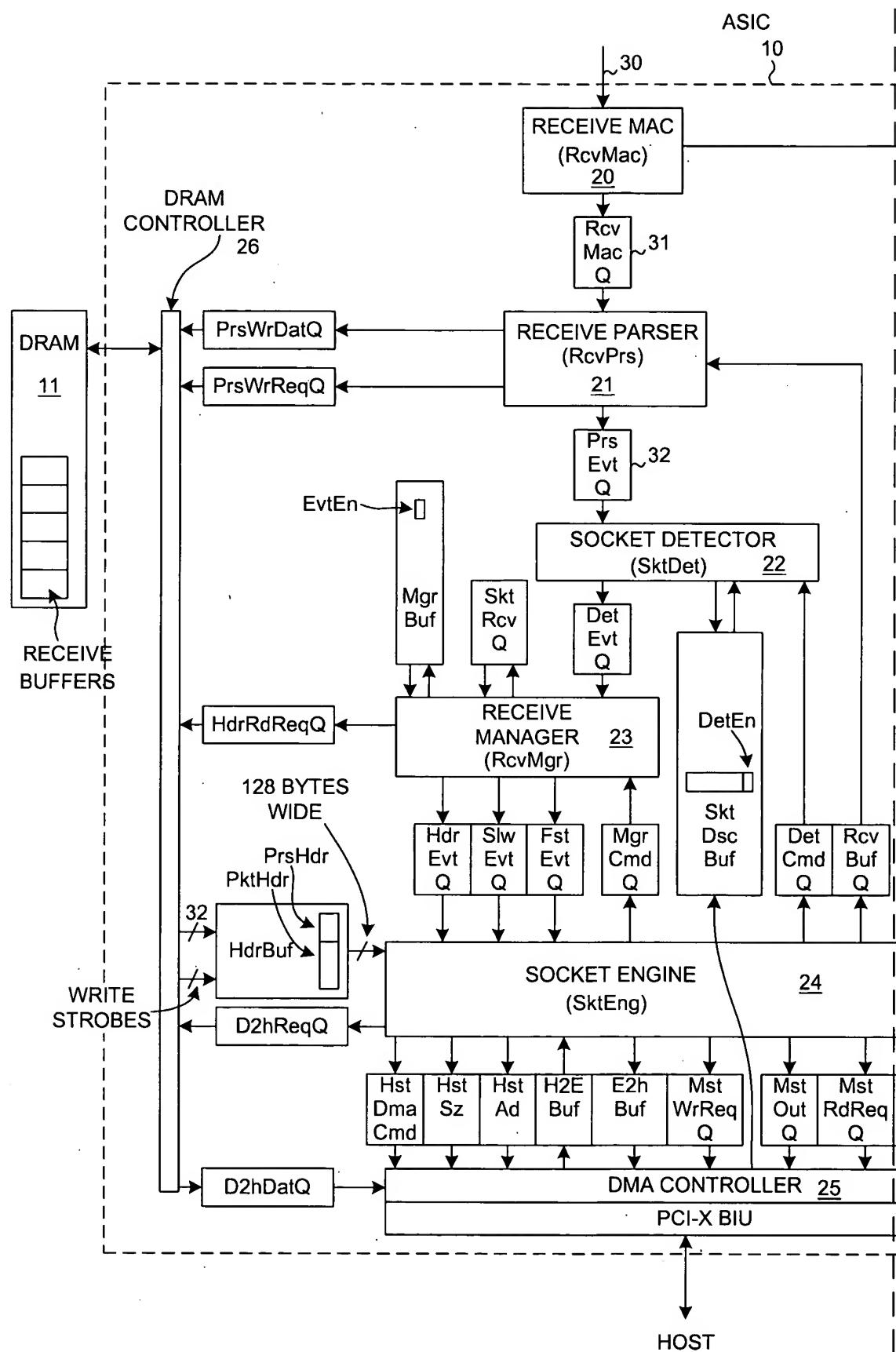
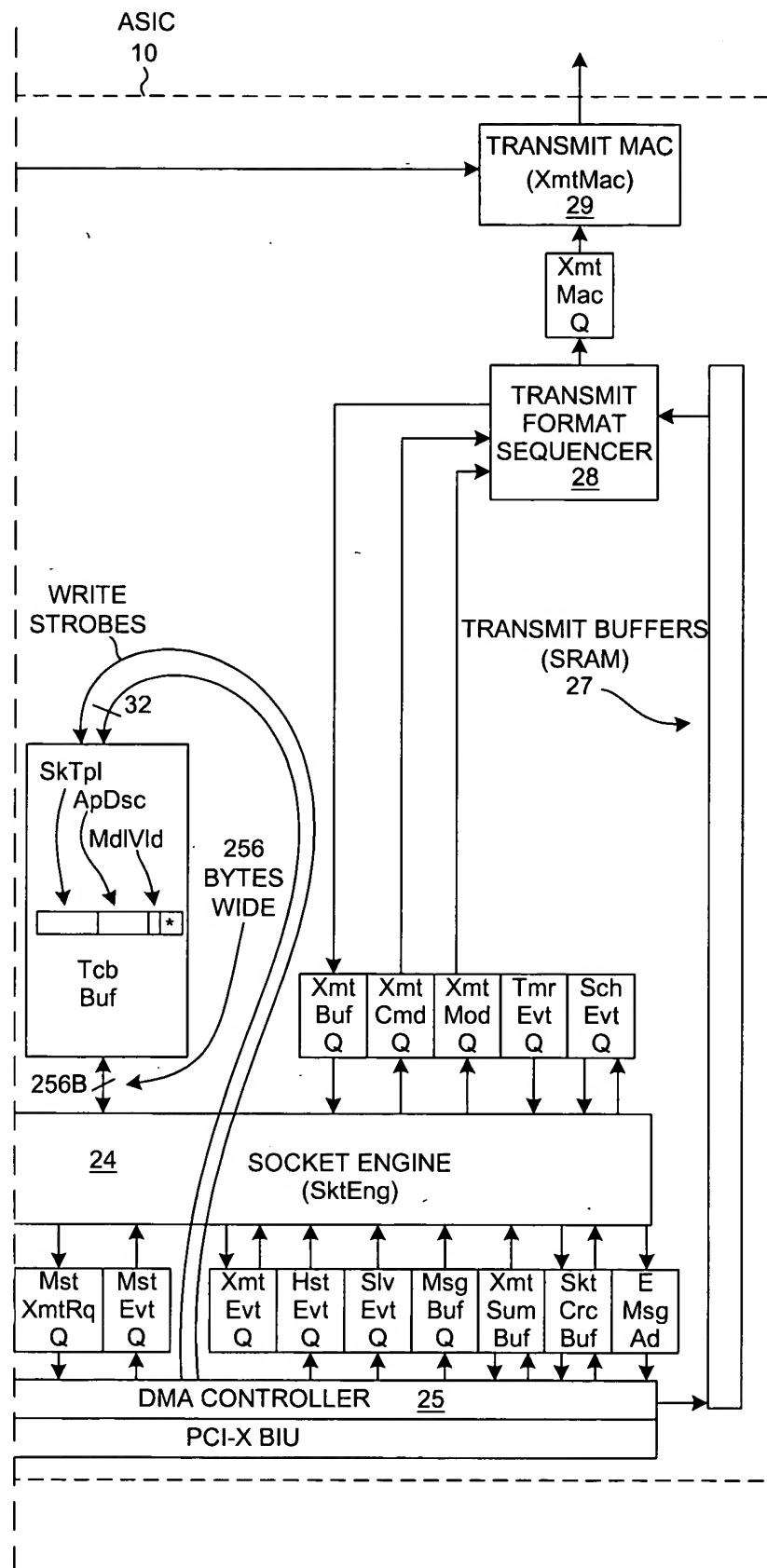


FIG. 2A



KEY TO FIG. 2

FIG. 2A	FIG. 2B
------------	------------

FIG. 2

FIG. 2B

4/24

BLOCK	DESCRIPTION
RcvMac	RECEIVE MEDIA ACCESS CONTROLLER
RcvPrs	RECEIVE PARSER.
SktDet	SOCKET DETECTOR.
RcvMgr	RECEIVE MANAGER.
SktEng	SOCKET ENGINE.

FIG. 3

MEMORY	FORMAT	DESCRIPTION
HdrBuf	[HdrId]{ PrsHd , PktHd }	HEADER BUFFER.
RcvMac	[BufId]{ PrsHd , RcvPk }	RECEIVE PACKET BUFFER.
SktDscBuf	[TcbId]{ SktDsc , DetEn }	SOCKET DESCRIPTOR BUFFER.
MgrBuf	[TcbId]{ EvtEn }	RECEIVE MANAGEMENT CONTROL.
TcbBuf	[TcbId]{ SktPl , MdIVd , ApDsc , * }	TCB BUFFER (ASTERISK INDICATES ADDITIONAL VALUES - SEE FIG. 20)
HstMem	[MsgAd]{ MsgHd , MsgDt } [CmdAd]{ CmdHd , CmdDt } [TcbAd]{ SktDsc , SktPl , SktHdr } [ApDsc]{ Data }	HOST MEMORY.
RcvBufQ	{BufId }	QUEUE OF FREE DRAM RECEIVE BUFFERS.
RcvMacQ	{RcvPk , RcvSt }	RECEIVE PACKET BUFFERING QUEUE.
PrsEvtQ	{PkEnd , SktHsh , SktDsc }	PARSE EVENT QUEUE.
DetCmdQ	{EnbCd , TcbId } {DbICd , TcbId }	SOCKET DETECTOR COMMAND QUEUE.
DetEvtQ	{EnbCd , TcbId } {DbICd , TcbId } {SlwCd , HdrCd , PkEnd } {FstCd , HdrCd , PkEnd , TcbId }	SOCKET DETECTOR EVENT QUEUE.
MgrCmdQ	{ArmCd , TcbId } {ClrCd , TcbId } {PshCd , TcbId } {ReqCd , TcbId }	RECEIVE MANAGER COMMAND QUEUE.
SktRcvQ	{HdrCd , PkEnd , HdrCd , PkEnd }	RECEIVE DESCRIPTOR QUEUES (ONE PER TCB).
SlwEvtQ	{EnbCd , TcbId } {DbICd , TcbId } {SlwCd , HdrCd , PkEnd }	SLOW RECEIVE QUEUE.
FstEvtQ	{ClrCd , TcbId } {FstCd , TcbId , HdrId }	FAST RECEIVE QUEUE.
HdrEvtQ	{TcbId }	NOTIFY SKT ENG OF HEADER IN HDR BUFFER.
HstEvtQ	{CmdAd , TcbId }	SOCKET ENGINE COMMAND QUEUE.
MsgBufQ	{MsgAd }	QUEUE OF FREE HOST MESSAGE BUFFERS.

FIG. 4

MNEMONIC	DESCRIPTION
EngCx	SOCKET ENGINE CONTEXT.
HdSel	CONTEXT HEADER BUFFER SELECT.
TcbId	TCB BUFFER IDENTIFIER.
BufId	RECEIVE BUFFER IDENTIFIER.
HdrId	HEADER BUFFER IDENTIFIER.
CmdAd	HOST COMMAND ADDRESS.
MsgAd	HOST MESSAGE ADDRESS.
TcbAd	ADDRESS OF TCB BUFFER ON THE HOST.
ApDsc	APPLICATION BUFFER ADDRESS (STORED IN TCB BUF).
MsgHd	MESSAGE HEADER (MESSAGES ARE SENT FROM NID TO HOST)(INCLUDES A TCBID).
CmdHd	COMMAND HEADER (COMMANDS ARE SENT FROM HOST TO NID).
MsgDt	MESSAGE DATA.
CmdDt	COMMAND DATA.
SkHdr	PROTOTYPE TRANSMIT HEADER.
LnkHd	LINK HEADER.
SnpHd	802.3 SNAP HEADER.
VldHd	VLAN HEADER.
NetHd	NETWORK HEADER
TptHd	TRANSPORT HEADER.
PayLd	PACKET PAYLOAD.
PktHd	PACKET HEADER.
RcvSt	RECEIVE MAC PACKET STATUS (GENERATED BY RECEIVE MAC).
NetIx	NETWORK HEADER START INDEX.
TptIx	TRANSPORT HEADER START INDEX.
Ddplx	DDP HEADER START INDEX.
PKLen	PACKET LENGTH.
PkEnd	RECEIVE BUFFER ENDING ADDRESS.
SkHsh	SOCKET HASH {TCB[N-1:03]}
PrsSt	PACKET PARSE STATUS.
DetEn	SOCKET DETECTION ENABLE BIT.
MdlVd	MEMORY DESCRIPTOR VALID (STORED IN TCB BUF).
HdrCd	HEADER LENGTH CODE.
SrcPt	TRANSPORT SOURCE PORT.
DstPt	TRANSPORT DESTINATION PORT.
SrcAd	NETWORK SOURCE ADDRESS.
DstAd	NETWORK DESTINATION ADDRESS.
EnbCd	RECEIVE ENABLE CODE (2-BITS).
DblCd	RECEIVE DISABLE CODE.
ArmCd	RECEIVE ARM CODE (SENT FROM SKT ENG TO RCV MGR).
ClrCd	RECEIVE DISARM CODE.
PshCd	RECEIVE PACKET RETURN CODE.
SlwCd	RECEIVE SLOW PATH CODE.
FstCd	RECEIVE FAST PATH CODE.
ReqCd	REQUEST CODE (TO TELL RCMGR TO PUT HEADERS INTO HEADER BUFFER).
EvtEn	RECEIVE EVENT ENABLE BIT (STORED IN MGR BUF).
SkTpl	SOCKET TUPLE.
RcvPk	RECEIVE PACKET.
PrsHd	PARSE HEADER (GENERATED BY RECEIVE PARSER).
SktDsc	SOCKET DESCRIPTOR.
EvtCd	EVENT CODE (FOR FstEvtQ IT CAN EITHER BE FstCd OR ClrCd). (FOR SlwEvtQ IT CAN EnbCd, DblCd, or SlwCd).

MNEMONIC	FORMAT
SkTpl	{ XmtAckNum, XmtSeqNum, XmtSeqLmt, XmtCcwSz, MaxSegSz, MaxXmtWin, RcvSeqLmt, ExpRcvSeq, ExpHdrLen, TcbAd }
HdrId	{ EngCx, HdSel }
TplUpd	{ PktAckNum, NxtSeqMax, NxtCcwSz, NxtExpSeq }
RcvPk	{ PktHd, PayLd }
PktHd	{ LnkHd, SnpHd, VlnHd, NetHd, TptHd }
PrsHd	{ PktBufId, HdrCd, PktPaySz, NetIx, TptIx, Ddplx, PkEnd, PrsSt * }
SktDsc	{ HdrCd, SrcPt, DstPt, SrcAd, DstAd, DetEn }
PkEnd	{ BufId, PkLen }
NetHd	{ PktSrcAddr, PktDstAddr, * }
TptHd	{ PktRcvSeq, PktXmtAck, PktXmtWin, PktSrcPrt, PktDstPrt, * }

* = INCLUDES OTHER VARIABLES NOT LISTED

FIG. 6

```

(700) RcvMac
(701) parse RcvPk
(702) write RcvMacQ      { RcvPk }           // FORWARD RECEIVE PACKET.
(703) write RcvMacQ      { RcvSt }           // APPEND STATUS.

(704) RcvPrs
(705) read  RcvBufQ      { BufId }           // GET A RECEIVE BUFFER.
(706) parse  RcvMacQ     { RcvPk , RcvSt }    // PARSE A PACKET.
(707) write  RcvBuf  [BufId] { PrsHd , RcvPk } // SAVE PACKET AND PARSE INFO.
(708) write  PrsEvtQ     { PkEnd , SkHsh , SktDsc } // SEND EVENT DESCRIPTOR.

(709) SktDet
(710) read  PrsEvtQ     { PkEnd , SkHsh , SktDsc } // GET EVENT DESCRIPTOR.
(711) test   SktDscBuf [SkHsh] { SktDsc ,..... SktDsc } , SktDsc // SEE IF FAST PATH.
(712) write  DetEvtQ     { SlwCd , HdrCd , PkEnd } // SEND SLOW DESCRIPTOR.

(713) RcvMgr
(714) read  DetEvtQ     { SlwCd , HdrCd , PkEnd } // GET SLOW DESCRIPTOR.
(715) write  SlwEvtQ     { SlwCd , HdrCd , PkEnd } // PASS TO SktEng.

(716) SktEng
(717) read  SlwEvtQ     { SlwCd , HdrCd , PkEnd } // GET SLOW DESCRIPTOR.
(718) read  MsgBufQ      { MsgAd }           // GET HOST BUFFER.
(719) copy   RcvBuf  [BufId] { PrsHd , RcvPk } , HstMem[MsgAd] // COPY PACKET.
(720) write  HstMem  [MsgAd] { MsgHd }           // NOTIFY HOST.
(721) write  RcvBufQ     { BufId }           // RECYCLE RECEIVE BUFFER.

```

SLOW PATH RECEIVE SEQUENCE

FIG. 7

FIRST PHASE (SLOW-PATH PURGE)

(800) Host

(801) write HstMem [CmdAd] { CmdHd , TcbAd } // WRITE RECEIVE COMMAND.

(802) write HstEvtQ { CmdAd , Tcbld } // SEND COMMAND NOTICE.

(803) SktEng

(804) read HstEvtQ { CmdAd , Tcbld } // GET NOTICE COMMAND.

(805) read HstMem [CmdAd] { CmdHd , TcbAd } // LOAD EnbSk COMMAND.

(806) copy HstMem [TcbAd] { SktDsc } , SktDscBuf [Tcbld] // LOAD SOCKET DESCRIPTOR.

(807) write DetCmdQ { EnbCd , Tcbld } // SEND DETECT ENABLE COMMAND.

(808) SktDet

(809) read DetCmdQ { EnbCd , Tcbld } // GET DETECT ENABLE COMMAND.

(810) set SktDscBuf [Tcbld] { DetEn } // ENABLE SOCKET DETECTION.

(811) write DetEvtQ { EnbCd , Tcbld } // SEND PURGE MARKER.

(812) RcvMgr

(813) read DetEvtQ { EnbCd , Tcbld } // GET PURGE MARKER.

(814) write SlwEvtQ { EnbCd , Tcbld } // SEND TO SOCKET ENGINE.

(815) SktEng

(816) read SlwEvtQ { EnbCd , Tcbld } // GET PURGE MARKER.

(817) read MsgBufQ { MsgAd } // GET A HOST MESSAGE BUFFER.

(818) write HstMem [MsgAd] { MsgHd } // NOTIFY HOST OF COMMAND DONE.
(THE MsgHd INDICATES AN ENABLE MARK MESSAGE EnbMrkMsg.)

SECOND PHASE (LOAD SOCKET STATE)

(819) Host

(820) read HstMem [MsgAd] { MsgHd } // GET MESSAGE THAT PRIOR COMMAND IS DONE.

(821) write HstMem [TcbAd] { SkTp } // WRITE SOCKET STATE.

(822) write HstMem [CmdAd] { CmdHd , TcbAd } // WRITE RECEIVE COMMAND.

(823) write HstEvtQ { CmdAd , Tcbld } // SEND COMMAND NOTICE.

(824) SktEng

(825) read HstEvtQ { CmdAd , Tcbld } // GET NOTICE COMMAND.

(826) read HstMem [CmdAd] { CmdHd , TcbAd } // LOAD HOST COMMAND.

(827) copy HstMem [TcbAd] { SkTp } , TcbBuf [Tcbld] // MOVE SOCKET STATE TO NID.

(828) write MgrCmdQ { ArmCd , Tcbld } // SEND ARM COMMAND TO RCV MGR.

(829) RcvMgr

(830) read MgrCmdQ { ArmCd , Tcbld } // GET ARM COMMAND.

(831) if (SktRcvQRdy[Tcbld])

(832) write FstEvtQ [TcbAd] { FstCd , Tcbld } // PUT FAST-PATH EVENT TO SKT ENG.

(833) else

(834) set MgrBuf [Tcbld] { EvtEn } // SET EVENT ENABLE BIT SO NEXT TIME A FAST-PATH DESCRIPTOR APPREARS ON SKT RCV QUEUE, IT WILL BE PASSED TO SOCKET ENGINE AS A FAST-PATH EVENT.

CONNECTION HANDOUT SEQUENCE

```

(900) RcvMac
(901) parse RcvPk
(902) write RcvMacQ { RcvPk } // FORWARD RECEIVE PACKET.
(903) write RcvMacQ { RcvSt } // APPEND STATUS.

(904) RcvPrs
(905) read RcvBufQ { Bufld } // GET A RECEIVE BUFFER.
(906) parse RcvMacQ { RcvPk, RcvSt } // PARSE A PACKET.
(907) write RcvBuf [Bufld] { PrsHd, RcvPk, SktDs } // SAVE PACKET AND PARSE INFO.
(908) write PrsEvtQ { PkEnd, SkHsh, SktDs } // SEND EVENT DESCRIPTOR.

(909) SktDet
(910) read PrsEvtQ { PkEnd, SkHsh, SktDs } // GET EVENT DESCRIPTOR.
(911) test SktDscBuf [SkHsh] { SktDs, ... } // SEE IF FAST-PATH FRAME.
(912) write DetEvtQ { FstCd, HdrCd, PkEnd } // SEND FAST DESCRIPTOR.
(913) write DetEvtQ { Tcbld, } c

(914) RcvMgr
(915) read DetEvtQ { FstCd, HdrCd, PkEnd } // GET FAST DESCRIPTOR.
(916) read DetEvtQ { Tcbld } c
(917) write SktRcvQ [Tcbld] { SlwCd, HdrCd, PkEnd } // SAVE TO SOCKET QUEUE.
(918) if (MgrBuf[Tcbld]{EvtEn}) begin
(919) write FstEvtQ { FstCd, Tcbld } // SEND FAST EVENT NOTICE.
(920) clr MgrBuf [Tcbld] { EvtEn } // CLR EVENT ENABLE.
(921) end

(922) SktEng
(923) read FstEvtQ { FstCd, Tcbld } // GET FAST EVENT NOTICE.
(924) write MgrCmdQ { ReqCd, HdrId, Tcbld } // REQUEST HEADER DELIVERY.

(925) RcvMgr
(926) read MgrCmdQ { ReqCd, HdrId, Tcbld } // GET HEADER REQUEST.
(927) read SktRcvQ [Tcbld] { SlwCd, HdrCd, PkEnd } // GET RCV DESCRIPTOR
(928) copy RcvBuf [Bufld] { PrsHd, PktHd } // GET FAST HEADERS.
(929) write HdrEvtQ { Tcbld, HdrBuf } // SEND HEADER EVENT.

(930) SktEng
(931) read HdrEvtQ { Tcbld } // GET HEADER EVENT.

(932) Check packet ack win and seq;
(933) test HdrBuf [HdrId] against TcbBuf[Tcbld] // CHECK ACK, WINDOW AND SEQ.

(934) if (TcbBuf [Tcbld]{MdlVd}) begin // IF VALID MEM DESCRIPTOR.
(935) read HdrBuf [HdrId] { Bufld } // GET SOURCE POINTER.
(936) read TcbBuf [Tcbld] { ApDsc } // GET DESTINATION POINTER.
(937) copy RcvBuf [Bufld] { PayLd } , HstMem[ApDsc] // MOVE FAST DATA.
(938) read MsgBufQ { MsgAd } // GET A HOST BUFFER.
(939) write HstMem [MsgAD] { MsgHd } // SEND RESPONSE MESSAGE.
(940) write RcvBufQ { Bufld } // RECYCLE MDL VALID BIT.
(941) clear TcbBuf [Tcbld] { MdlVd } // CLEAR MDL VALID BIT.
(942) end

```

FAST-PATH RECEIVE SEQUENCE

FIG. 9A

```

(943) else begin                                // IF NO VALID MEMORY DESCRIPTOR, SEND INITIAL DATA.
(944)   read   MsgBufQ      { MsgAd }           // GET A HOST BUFFER.
(945)   copy   RcvBuf  [BufId] { PayLd }        // COPY FAST DATA.
(946)   write  HstMem [MsgAd] { MsgHd } , HstMem[MsgAd] // REQUEST MEMORY DESCRIPTOR.
(947)   write  RcvBufQ      { BufId }           // RECYCLE RECEIVE BUFFER.

(948) Host
(949)   read   HstMem [MsgAd] { MsgHd , MsgDt } // GET RECEIVE REQUEST.
(950)   write  HstMem [CmdAd] { CmdHd , ApDsc } // WRITE RECEIVE COMMAND.
(951)   write  HstEvtQ      { CmdAd , TcbId }   // SEND COMMAND NOTICE.

(952) SktEng
(953)   read   HstEvtQ      { CmdAd , TcbId }   // GET COMMAND NOTICE.
(954)   copy   HstMem [CmdAd] { CmdHd , ApDsc } , TcbBuf [TcbId] // LOAD RECEIVE COMMAND.
(955)   set    TcbBuf  [TcbId] { MdlVd , }        // SET VALID BIT.
(956)   write  MgrCmdQ     { ArmCd , TcbId }   // LOAD RECEIVE COMMAND.
(957) RcvMgr
(958)   read   MgrCmdQ     { ArmCd , TcbId }   // GET ARM COMMAND.
(959)   if (SktRcvQRdy [TcbId])
(960)     write FstEvtQ     { FstCd , TcbId }   // SEND NOTICE TO SKT ENG.
(961) end

```

FAST-PATH RECEIVE SEQUENCE (CONTINUED)

FIG. 9B

KEY TO FIG. 9

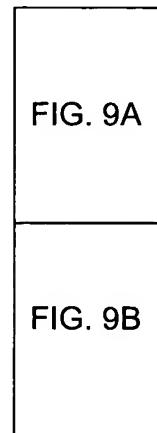


FIG. 9

10/24

```

(1000) Idle: if (SlwEvtQRdy) begin           // SERVICE SLOW EVENT QUEUE.
(1001)     if (SlwEvtQ {EvtCd} == 0) begin    // SLOW RECEIVE EVENT.
(1002)         EState    <= SlwRcvEvt;        // GO TO EVENT SERVICE.
(1003)         ETcbId   <= SlwEvtQ{TcbId};  // SAVE TCB NUMBER.
(1004)         ECmdAd   <= x;                //
(1005)         EHdrCd   <= SlwEvtQ{HdrCd}; // SAVE HEADER DMA LENGTH.
(1006)         EHdrAd   <= x;                //
(1007)         EBuflId  <= SlwEvtQ{BufId}; // SAVE RECEIVE BUFFER NUMBER.
(1008)         EPkLen   <= SlwEvtQ{PkLen}; // SAVE RECEIVE BUFFER LENGTH.
(1009)     end

(1010)     else if (SlwEvtQ {EvtCd} == 1) begin // SLOW MARK EVENT.
(1011)         EState    <= EnbMrkEvt;        // GO TO EVENT SERVICE.
(1012)         ETcbId   <= SlwEvtQ{TcbId};  // SAVE TCB NUMBER.
(1013)         EcmdAd   <= x;                //
(1014)         EHdrCd   <= x;                //
(1015)         EHdrAd   <= x;                //
(1016)         EBuflId  <= x;                //
(1017)         EPkLen   <= x;                //
(1018)     end

(1019)     else begin                         // DISABLE MARK EVENT.
(1020)         EState    <= DblMrkEvt;        // GO TO EVENT SERVICE.
(1021)         ETcbId   <= SlwEvtQ{TcbId};  // SAVE TCB NUMBER.
(1022)         EcmdAd   <= x;                //
(1023)         EHdrCd   <= SlwEvtQ{HdrCd}; // SAVE HEADER DMA LENGTH.
(1024)         EHdrAd   <= x;                //
(1025)         EBuflId  <= SlwEvtQ{BufId}; // SAVE RECEIVE BUFFER NUMBER.
(1026)         EPkLen   <= SlwEvtQ{PkLen}; // SAVE RECEIVE BUFFER LENGTH.
(1027)     end
(1028) end

(1029) else if (FstEvtQRdy) begin           // SERVICE FAST EVENT QUEUE.
(1030)     if (FstEvtQ {EvtCd} == 0) begin    // FAST RECEIVE EVENT (EvtCd is the FstCd).
(1031)         EState    <= FstRcvEvt;        // GO TO EVENT SERVICE.
(1032)         ETcbId   <= FstEvtQ{TcbId};  // SAVE TCB NUMBER.
(1033)         EcmdAd   <= x;                //
(1034)         EHdrCd   <= x;                //
(1035)         EHdrAd   <= FstEvtQ{HdrId}; // SAVE HEADER BUFFER POINTER.
(1036)         EBuflId  <= x;                //
(1037)         EPkLen   <= x;                //
(1038)     end

(1039)     else begin                         // CLEAR MARK EVENT (EvtCd is the ClrCd).
(1040)         EState    <= ClrMrkEvt;        // GO TO EVENT SERVICE.
(1041)         ETcbId   <= FstEvtQ{TcbId};  // SAVE TCB NUMBER.
(1042)         EcmdAd   <= x;                //
(1043)         EHdrCd   <= x;                //
(1044)         EHdrAd   <= FstEvtQ{HdrId}; // SAVE HEADER BUFFER POINTER.
(1045)         EBuflId  <= x;                //
(1046)         EPkLen   <= x;                //
(1047)     end
(1048) end

```

SOCKET ENGINE STATES
FIG. 10A

```

(1049) else if (HstEvtQRdy).begin           // SERVICE HOST EVENT QUEUE.
(1050)   EState    <= SktCmdEvt;           // GO TO EVENT SERVICE.
(1051)   ETcbId   <= HstEvtQ{TcbId};     // SAVE TCB NUMBER.
(1052)   EcmdAd   <= HstEvtQ{CmdAd};     // SAVE COMMAND BLOCK ADDRESS.
(1053)   EHdrCd   <= x;                  //
(1054)   EHdrAd   <= x;                  //
(1055)   EBuflId  <= x;                  //
(1056)   EPkLen   <= x;                  //
(1057) end

(1058) else if (HdrEvtQRdy).begin          // SERVICE HEADER EVENT QUEUE.
(1059)   EState    <= HdrDmaEvt;           // GO TO EVENT SERVICE.
(1060)   ETcbId   <= HdrEvtQ{TcbId};     // SAVE TCB NUMBER IN ETcbId (MAKES
(1061)   EcmdAd   <= x;                  ALL THE BITS OF THE TCB SIMULTANEOUSLY
(1062)   EHdrCd   <= x;                  AVAILABLE TO THE SKT ENG.)
(1063)   EHdrAd   <= HdrEvtQ{HdrId};     // SAVE HEADER BUFFER POINTER IN EHdrAd
(1064)   EBuflId  <= x;                  (MAKES ALL THE BITS OF THE HEADER BUFFER
(1065)   EPkLen   <= x;                  SIMULTANEOUSLY AVAILABLE TO THE SKT ENG.)
(1066) end

(1067) else begin                         // NO EVENT TO SERVICE.
(1068)   EState    <= Idle;                // KEEP CHECKING FOR WORK.
(1069)   ETcbId   <= HdrEvtQ{TcbId};     //
(1070)   EcmdAd   <= x;                  //
(1071)   EHdrCd   <= x;                  //
(1072)   EHdrAd   <= x;                  //
(1073)   EBuflId  <= x;                  //
(1074)   EPkLen   <= x;                  //
(1075) end

(1076) // Slow Path Receive Event.
(1077) SlwRcvEvt: begin                  // DMA SLOW-PATH PACKET TO HOST.
(1078)   EState    <= SlwRcv0;           // SET NEXT STATE.
(1079)   EMsgAd   <= MsgBufQ{MsgAd};    // GET HOST BUFFER.
(1080)   DrmAd    <= EBuflId<<11;      // DRAM SOURCE ADDRESS.
(1081)   HstAd    <= MsgBufQ{MsgAd} + MsgHdLen; // HOST DESTINATION ADDRESS.
(1082)   HstSz    <= EPkLen;             // DMA LENGTH.
(1083)   HstDmaCmd <= R2hCd;            // MOVE RCV BUFFER TO HOST DMA.
(1084) end

(1085) SlwRcv0: begin                  // SEND HOST NOTIFICATION MESSAGE.
(1086)   EState    <= Idle;              // GO FIND WORK.
(1087)   E2hBuf   <= SlwRcvMsg;         // SET UP SLOW RECEIVE MESSAGE.
(1088)   HstAd    <= EMsgAd;            // HOST DESTINATION ADDRESS.
(1089)   HstSz    <= MsgHdLen;           // DMA LENGTH.
(1090)   HstDmaCmd <= E2hCd;            // MOVE MESSAGE TO HOST.
(1091)   RcvBufQ  <= EBuflId;           // RECYCLE RECEIVE BUFFER.
(1092) end

```

SOCKET ENGINE STATES (CONTINUED)

FIG. 10B

12/24

```

(1093) // Slow-Path Purge Event (Socket Detection Enabled).
(1094)  EnbMrkEvt: begin                                // NOTIFY HOST FIRST PHASE OF HANDOUT IS DONE.
(1095)    EState      <= Idle;                          // GO FIND WORK.
(1096)    EMsgAd     <= MsgBufQ{MsgAd};               // GET HOST MESSAGE BUFFER ADDRESS.
(1097)    E2hBuf      <= EnbMrkMsg;                   // PREPARE AN ENABLE-MARK MESSAGE.
(1098)    HstAd       <= MsgBufQ{MsgAd};               // SET HOST DESTINATION ADDRESS.
(1099)    HstSz       <= MsgHdLen;                   // SET DMA LENGTH.
(1100)    HstDmaCmd  <= E2hCd;                        // MOVE MESSAGE TO HOST MESSAGE BUFFER.
(1101)  end

(1102) // Descriptor Buffer Release Event (Socket Detection Disabled).
(1103)  DblMrkEvt: begin                                // DISABLE MARK EVENT.
(1104)    EState      <= DblMrk0;                      // GO TO DISABLE SERVICE.
(1105)    EMsgAd     <= MsgBufQ{MsgAd};               // GET HOST BUFFER.
(1106)  end

(1107)  DblMrk0: begin                                 // NOTIFY HOST.
(1108)    EState      <= Idle;                          // GO FIND WORK.
(1109)    E2hBuf      <= DblMrkMsg;                   // DISABLE-MARK MESSAGE.
(1110)    HstAd       <= EMsgAd;                      // HOST DESTINATION ADDRESS.
(1111)    HstSz       <= MsgHdLen;                   // DMA LENGTH.
(1112)    HstDmaCmd  <= E2hCd;                        // DO NID TO HOST DMA.
(1113)  end

(1114) // Fast-Path Receive Event.
(1115)  FstRcvEvt: begin                                // GET FAST PACKET HEADER.
(1116)    EState      <= Idle;                          // GO FIND WORK.
(1117)    MgrCmdQ   <= {ReqCd, PktId, ETcbId}        // REQUEST HEADER DELIVERY.
(1118)  end

(1119) // Fast-Receive Purge Event (Fast Event Disabled).
(1120)  ClrMrkEvt: begin
(1121)    EState      <= ClrMrk0;                      // GO TO NEXT STATE.
(1122)    EMsgAd     <= MsgBufQ{MsgAd};               // GET MESSAGE BUFFER ON HOST.
(1123)    HstTcbId   <= ETcbId;                        // SOURCE IS TCB BUFFER.
(1124)    HstAd       <= TcbBuf{TcbAd};                // DESTINATION ON HOST.
(1125)    HstSz       <= SkTpLen;                      // DMA LENGTH.
(1126)    HstDmaCmd  <= T2hCd;                        // MOVE TCB FROM NID TO HOST.
(1127)    DetCmdQ   <= {DblCd, ETcbId};               // SEND DISABLE COMMAND TO SKT DET.
(1128)  end

(1129)  ClrMrk0: begin                                 // NOTIFY HOST THAT STATE HAS BEEN EXPORTED.
(1130)    EState      <= Idle;                          // GO FIND WORK.
(1131)    E2hBuf      <= ExportMsg;                   // STATE EXPORT MESSAGE INTO E2HBUF.
(1132)    HstAd       <= EMsgAd;                      // DESTINATION ADDRESS ON HOST.
(1133)    HstSz       <= MsgHdLen;                   // DMA LENGTH.
(1134)    HstDmaCmd  <= E2hCd;                        // MOVE MESSAGE FROM NID TO HOST.
(1135)  end

(1136) // Host Command Entry Event.
(1137)  SktCmdEvt: begin                                // GET COMMAND FROM HOST.
(1138)    EState      <= SktCmd0;                      // SET NEXT STATE.
(1139)    HstAd       <= ECmdAd;                      // SET HOST SOURCE ADDRESS.
(1140)    HstSz       <= CmdHdLen;                   // SET DMA LENGTH.
(1141)    HstDma Cmd <= H2eCd;                        // MOVE COMMAND FROM HOST TO NID.
(1142)  end

```

SOCKET ENGINE STATES (CONTINUED)

FIG. 10C

```

(1143) SktCmd0: if (H2eBuf {CmdCd} == 0) begin // IF ENABLE COMMAND.
(1144)   EState    <= SktEnbCmd;           // GO TO ENABLE ROUTINE.
(1145) end
(1146) else if (H2eBuf {CmdCd} == 1) begin // IF ARM COMMAND.
(1147)   EState    <= SktArmCmd;           // GO TO ARM ROUTINE.
(1148) end
(1149) else begin                         // MUST BE RCV COMMAND.
(1150)   EState    <= SktRcvCmd;           // GO TO ARM ROUTINE.
(1151) end

(1152) // Socket Enable Command Service.
(1153) SktEnbCmd: begin                  // GET SOCKET DESCRIPTOR.
(1154)   EState    <= SktEnb0;             // GO TO NEXT STATE.
(1155)   DscBufAd  <= ETcbl0 * SktDscLen; // ADDR FOR SktDsc BUFFER.
(1156)   HstAd     <= H2eBuf {TcbAd} + SktDsclx; // HOST SktDsc ADDRESS.
(1157)   HstSz     <= SktDscLen;           // DMA LENGTH.
(1158)   HstDmaCmd <= H2tCd;              // MOVE SkDsc FROM HOST TO NID.
(1159) end

(1160) SktEnb0: begin                  // ENABLE SOCKET DETECTION.
(1161)   EState    <= Idle;               // GO FIND WORK.
(1162)   DetCmdQ   <= {EnbCd, ETcbl0}; // SEND ENABLE COMMAND.
(1163) end

(1164) // Socket Arm Command Service.
(1165) SktArmCmd: begin                // GET SOCKET STATE.
(1166)   EState    <= SktArm0;            // GO TO NEXT STATE.
(1167)   TcbBufAd  <= ETcbl0 * TcbBufLen; // ADDR FOR SktDsc BUFFER.
(1168)   HstAd     <= H2eBuf {TcbAd} + SkTpilx; // HOST SkTpil ADDRESS.
(1169)   HstSz     <= SkTpilLen;           // DMA LENGTH.
(1170)   HstDmaCmd <= H2tCd;              // MOVE SkTpil FROM HOST TO NID.
(1171) end

(1172) SktArm0: begin                  // ENABLE SOCKET RECEIVE.
(1173)   EState    <= Idle;               // GO FIND WORK.
(1174)   MgrCmdQ   <= ArmCd, ETcbl0; // SEND ARM RECEIVE COMMAND.
(1175) end

(1176) // Socket Receive Command Service.
(1177) SktRcvCmd: begin                // GET APPLICATION BUFFER DSC.
(1178)   EState    <= SktRcv0;            // GO TO NEXT STATE.
(1179)   TcbBufAd  <= (ETcbl0 * TcbBufLen) + ApDsclx; // ADDR FOR APP DSC BUFFER.
(1180)   HstAd     <= ECmdAd + ApDsclx; // HOST ApDsc ADDRESS.
(1181)   HstSz     <= ApDscLen;           // DMA LENGTH.
(1182)   HstDmaCmd <= H2tCd;              // MOVE ApDsc FROM HOST TO NID.
(1183) end

(1184) SktRcv0: begin                  // ENABLE SOCKET RECEIVE.
(1185)   EState    <= Idle;               // GO FIND WORK.
(1186)   TcbBuf{RSqMx} <= TcbBuf{RSqMx}+H2eBuf{Sqlnc}; // INCREMENT RECEIVE WINDOW.
(1187)   MgrCmdQ   <= {ArmCd, ETcbl0}; // SEND ARM RECEIVE COMMAND.
(1188)   MdlVd[ETcbl0] <= 1;              // ADDR FOR MDL VALID BIT.
(1189) end

```

SOCKET ENGINE STATES (CONTINUED)

FIG. 10D

```

(1190) // Header Event Service.
(1191) HdrDmaEvt: begin
(1192)     EState      <= HdrEvt0;           // GO TO NEXT STATE.
(1193)     EMsgAd     <= MsgBufQ {MsgAd}; // GET HOST BUFFER.
(1194)     EFlush      <= FlushDet;        // FLUSH DETECT - PARALLEL OPERATION
(1195)     EBuflId    <= HdrBuf{BufId}; // SAVE RECEIVE BUFFER NUMBER.
(1196)     EPkLen     <= HdrBufQ{PkLen}; // SAVE PACKET LENGTH.
(1197) end

(1198) HdrEvt0: if (EFlush) begin           // IF SOCKET RECEIVE ERROR.
(1199)     EState      <= Idle;           // GO FIND WORK.
(1200)     MgrCmdQ    <= {PshCd, ETcblId}; // RETURN PACKET AND FLUSH.
(1201) end

(1202)     else if (MdlVd[ETcblId]) begin    // IF APP BUFFER DSCR IS VALID.
(1203)         EState      <= FastRcv;        // GO TO NEXT STATE.
(1204)         DrmAd      <= (EBuflId<<11)+PkHdrLen; // DRAM SOURCE ADDRESS.
(1205)         HstAd      <= TcbBuf{ApDsc}; // HOST DESTINATION ADDRESS.
(1206)         HstSz      <= EPkLen - HdrLen; // DMA LENGTH.
(1207)         HstDmaCmd <= R2hCd;          // DO RCV TO HOST DMA.
(1208)     end

(1209)     else                      // SEND FIRST FAST-PATH PACKET TO HOST.
(1210)         EState      <= InitRcv;        // SET NEXT STATE.
(1211)         DrmAd      <= EBuflId<<11; // SET DRAM SOURCE ADDRESS.
(1212)         HstAd      <= EMsgAd + MsgHdLen; // SET HOST DESTINATION ADDRESS.
(1213)         HstSz      <= EPkLen;          // SET DMA LENGTH.
(1214)         HstDmaCmd <= R2hCd;          // MOVE PACKET TO HOST.
(1215)         RcvBufQ   <= EBuflId;         // RECYCLE RECEIVE BUFFER.
(1216)     end

(1217) FastRcv: begin                  // NOTIFY HOST.
(1218)     EState      <= UpdMdlEntries; // 
(1219)     E2hBuf      <= FstRcvMsg;      // SEND FAST-PATH RECEIVE MESSAGE.
(1220)     HstAd      <= EMsgAd;          // SET HOST DESTINATION ADDRESS.
(1221)     HstSz      <= MsgHdLen;        // SET DMA LENGTH.
(1222)     HstDmaCmd <= E2hCd;          // MOVE MESSAGE TO HOST.
(1223)     RcvBufQ   <= EBuflId;         // RECYCLE RECEIVE BUFFER.
(1224)     TcbBuf      <= TplUpd;          // UPDATE SOCKET STATE - PARALLEL OPERATION.
(1225)     MgrCmdQ    <= {ArmCd, ETcblId}; // SEND ARM RECEIVE COMMAND TO RCV MGR.
(1226) end

(1227) UpdMdlEntries:
(1228)     if exhausted begin           // IF THE MDL ENTRY IS EXHAUSTED
(1229)         clear TcbBuf[TcblId]{MdlVd}; // CLEAR THE MDL VALID BIT
(1230)     end
(1231)     EState      <= FastRcv;        // GO TO THE IDLE STATE

(1232) InitRcv: begin // NOTIFY HOST THAT FIRST FAST-PATH PACKET IS IN MESSAGE BUFFER.
(1233)     EState      <= Idle;           // GO FIND WORK.
(1234)     E2hBuf      <= RcvReqMsg;      // FORM RECEIVE REQUEST MESSAGE.
(1235)     HstAd      <= EMsgAd;          // SET HOST DESTINATION ADDRESS.
(1236)     HstSz      <= MsgHdLen;        // SET DMA LENGTH.
(1237)     HstDmaCmd <= E2hCd;          // MOVE MESSAGE TO HOST.
(1238)     RcvBufQ   <= EBuflId;         // RECYCLE RECEIVE BUFFER.
(1239)     TcbBuf      <= TplUpd;          // UPDATE SOCKET STATE - PARALLEL OPERATION.
(1240) end

```

SOCKET ENGINE STATES (CONTINUED)

FIG. 10E

KEY TO FIG. 10

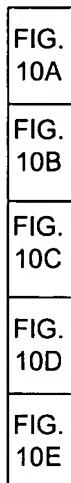


FIG. 10

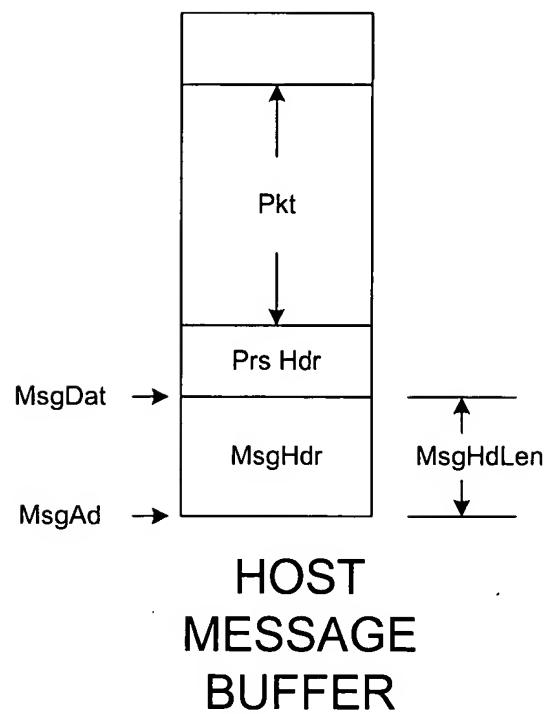


FIG. 11

FIRST PHASE (FAST-PATH EVENT PURGE)

- (1500) SktEng
- (1501) write MgrCmdQ { PshCd , Tcbld } // SEND DISARM COMMAND.
- (1502) RcvMgr
- (1503) read MgrCmdQ { PshCd , Tcbld } // GET DISARM COMMAND.
- (1504) write FstEvtQ { ClrCd , Tcbld } // SEND PURGE MARKER.
- (1505) clear MgrBuf [Tcbld] { EvtEn } // DISARM SOCKET RECEIVE.

SECOND PHASE (SOCKET STATE SAVE)

- (1506) SktEng
- (1507) read FstEvtQ { ClrCd , Tcbld } // GET DISARM MARKER.
- (1508) copy TcbBuf [Tcbld] { SkTpI } , HstMem[TcbAd] // SEND STATE TO HOST.
(ALSO SEE LINES 1123-1126).
- (1509) read MsgBufQ { MsgAd } // GET MESSAGE BUFFER.
- (1510) write HstMem [MsgAd] { MsgHd } // NOTIFY HOST.

THIRD PHASE (FAST-PATH QUEUE PURGE)

- (1511) SktEng
- (1512) write DetCmdQ { DblCd , Tcbld } // SEND DISABLE COMMAND.
- (1513) SktDet
- (1514) read DetCmdQ { DblCd , Tcbld } // GET DISABLE COMMAND.
- (1515) clr SktDscBuf [Tcbld] { DetEn } // DISABLE SOCKET DETECTION.
- (1516) RcvMgr
- (1517) read DetCmdQ { DblCd , Tcbld } // GET PURGE MARKER.
- (1518) while (SktRcvQRdy[Tcbld]) // IF SOCKET RCV DESCRIPTORS.
- (1519) copy DetEvtQ { SlwCd , HdrCd , PkEnd } , SlwEvtQ // MOVE DESCRIPTOR TO SLOW QUEUE.
- (1520) write SlwEvtQ { DblCd , Tcbld } // SEND PURGE MARKER.
- (1521) SktEng
- (1522) read SlwEvtQ { DblCd , Tcbld } // GET PURGE MARKER.

CONNECTION FLUSH SEQUENCE

FIG. 12

```

(1600) XmtWindow = XmtSeqLmt - XmtAckNum ; // TRANSMIT WINDOW.
(1601) PktAckSz = PktXmtAck - XmtAckNum ; // DATA BEING ACKED.
(1602) XmtUnAckd = XmtSeqNum - XmtAckNum ; // DATA TO BE ACKED.
(1603) XmtWinAvl = XmtSeqLmt - XmtSeqNum ; // AVAILABLE WINDOW.
(1604) CurCcwInc = XmtCcwSz + (MaxSegSz<<1) ; // NEW CONGESTION WINDOW.
(1605) NxtUnAckd = XmtSeqNum - PktXmtAck ; // DATA TO BE ACKED.
(1606) NxtXmtLmt = PktXmtAck + PckXmtWin ; // TRANSMIT LIMIT.
(1607) NxtWinAvl = NxtXmtLmt - XmtSeqNum ; // AVAILABLE WINDOW.
(1608) XmtAckNew = PktXmtAck != XmtAckNum ; // NEW ACK DETECT.
(1609) XmtWinNew = PktXmtWin != XmtWindow ; // WINDOW CHANGE.

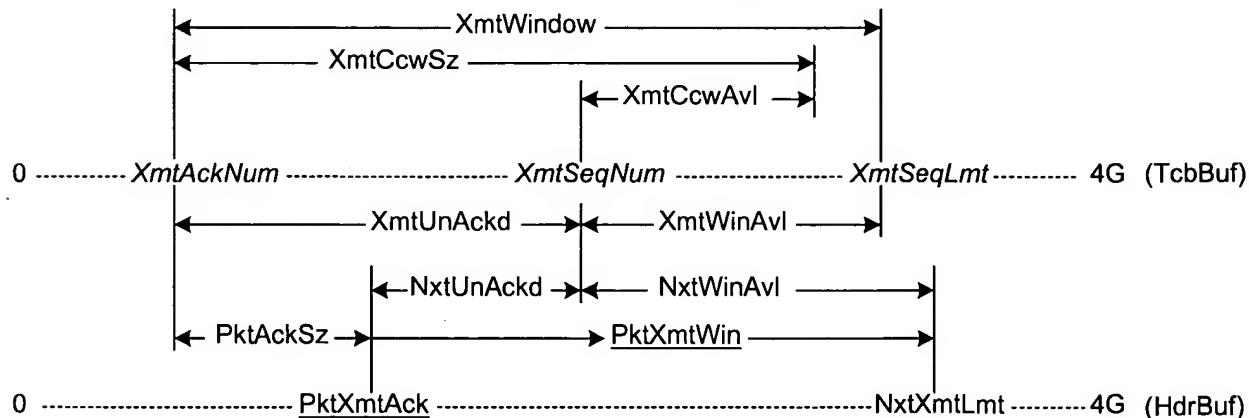
(1610) XmtAckDup = !XmtAckNew & !XmtWinNew ; // ACK IS DUPLICATE.
(1611) XmtAckVld = PktAckSz <= XmtUnAckd ; // ACK IS VALID.
(1612) XmtAckOld = PktAckSz > XmtWindow ; // OLD ACK DETECT.
(1613) CurCcwStp = XmtCcwSz < NxtXmtSz ; // CONGESTION WINDOW STOP.
(1614) CurWinStp = XmtWinAvl < NxtXmtSz ; // WINDOW STOP.
(1615) NxtSlwDet = CurCcwInc < MaxXmtWin ; // SLOW START DETECT.
(1616) NxtWinGrw = PktXmtWin > MaxXmtWin ; // XMT WINDOW IS GROWING.
(1617) NxtWinOpn = (XmtSeqLmt - NxtXmtLmt) !< 'Quadrant' ; // XMT WINDOW IS OPENING.
(1618) NxtWinShr = (NxtXmtLmt - XmtSeqLmt) !< 'Quadrant' ; // XMT WINDOW IS SHRINKING.
(1619) NxtWinStp = NxtWinAvl < NxtXmtSz ; // WINDOW STOP.
(1620) NxtXmtCcw = !XmtAckNew ? XmtCcwSz ; // NEXT CONGESTION CTRL WIN.
(1621) : NxtSlwDet ? CurCcwInc : MaxXmtWin ; // CONGESTION WINDOW STOP.
(1622) NxtCcwStp = !XmtAckNew ? ; // CONGESTION WINDOW STOP.
(1623) = ((XmtCcwSz - NxtUnAckd) << NxtXmtSz)
(1624) : NxtSlwDet ?
(1625) = ((CurCcwInc - NxtUnAckd) << NxtXmtSz)
(1626) : ((MaxXmtWin - NxtUnAckd) << NxtXmtSz)
(1627) XmtAckEvt = XmtAckNew & (PktXmtAck == XmtSeqNum); // ACKING ALL SENT DATA.
(1628) XmtWinEvt = !NxtWinStp & (CurWinStp | CurCcwStp) // TRANSMIT THRESHOLD DETECT.
(1629) & !NxtCcwStp;
```

ITALICS = INDICATES VALUE FROM TCB BUFFER

UNDERLINE = INDICATES VALUE FROM HEADER BUFFER

INCOMING ACK PROCESSING

FIG. 13



ITALICS = INDICATES VALUE FROM TCB BUFFER

UNDERLINE = INDICATES VALUE FROM HEADER BUFFER

INCOMING ACK PROCESSING

FIG. 14

```

(1700) CurWinFul = (ExpRcvSeq == RcvSeqLmt) ; //
(1701) HdrLenOk = (PktHdrLen == ExpHdrLen) ; //
(1702) PurAckDet = (PktPaySz == 0) ; //

(1703) ExpSeqDet = (PktRcvSeq == ExpRcvSeq); ; //

(1704) OldSeqDet = ((RcvSeqLmt - PktRcvSeq) < 'Quadrant) ; //
(1705) | (PktRcvSeq - ExpRcvSeq) < 'Quadrant) ; //

(1706) WinPrbDet = (CurWinFul & PktPaySz == 1) ; //
(1707) NxtSeqExp = (PktRcvSeq + PktPaySz) ; //

(1708) CurWinOvr = ((RcvSeqLmt - NxtSeqExp) < 'Quadrant) ; //
(1709) NewDatDet = ((NxtSeqExp - ExpRcvSeq) << 'Quadrant) ; //
(1710) & !OldSeqDet ; //
(1711) CurPktFul = (PktPaySz == MaxSegSz) ; //

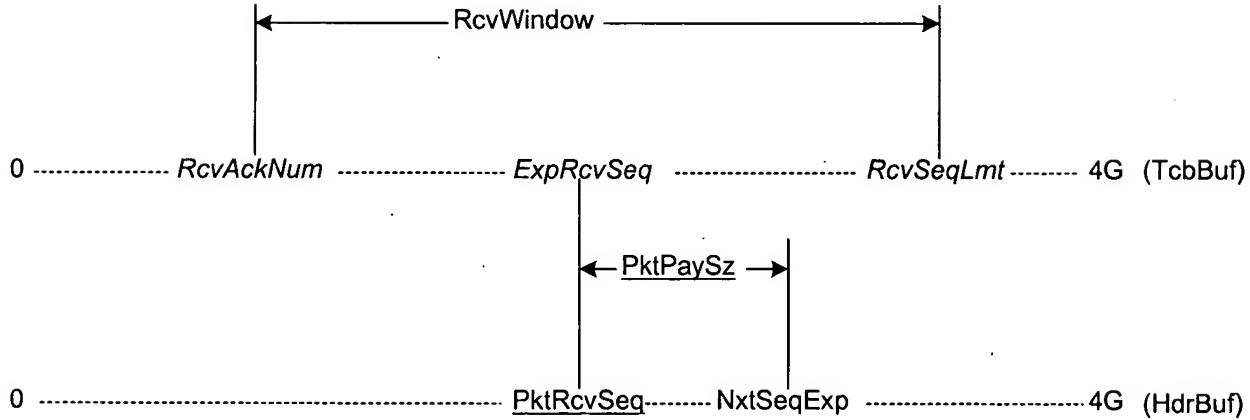
(1712) FlushDet = CurWinOvr | (!ExpSeqDet & !OldSeqDet) ; //
(1713) | NxtWinShr | (!XmtAckVld & !XmtAckOld) ;

```

ITALICS = INDICATES VALUE FROM TCB BUFFER
 UNDERLINE = INDICATES VALUE FROM HEADER BUFFER

INCOMING DATA PROCESSING

FIG. 15



ITALICS = INDICATES VALUE FROM TCB BUFFER
 UNDERLINE = INDICATES VALUE FROM HEADER BUFFER

INCOMING DATA PROCESSING

FIG. 16

0	7 8	15 16	23 24	31
FRAME STATUS A	FRAME STATUS A	FRAME STATUS A	FRAME STATUS A	
FRAME STATUS B	FRAME STATUS B	FRAME STATUS B	FRAME STATUS B	
TIME STAMP	TIME STAMP	TIME STAMP	TIME STAMP	
TIME ECHO	TIME ECHO	TIME ECHO	TIME ECHO	
SEQUENCE	SEQUENCE	SEQUENCE	SEQUENCE	
ACK	ACK	ACK	ACK	
WINDOW	WINDOW	PAYLOAD START	PAYLOAD START	
PAYLOAD LENGTH	PAYLOAD LENGTH	TCP CHECKSUM	TCP CHECKSUM	

HEADER BUFFER FORMAT

THE HEADER BUFFER IS REPRESENTED HERE AS MULTIPLE 32-BIT VALUES TO MAKE THE ILLUSTRATION MORE COMPACT FOR ILLUSTRATIVE PURPOSES. IN ACTUALITY, THE 32-BIT VALUES SET FORTH ABOVE ARE CONCATENATED END-TO-END. THE HEADER BUFFER IS ONE-BIT DEEP AND 8X32 BITS LONG.

FIG. 17

BIT	NAME	DESCRIPTION
31	802.3Oflw	The 802.3 size/count exhausted before the end of the frame.
30	TprtFlags	The transport flags require attention.
29	TprtOpt	Transport header options were detected.
28	TprtOflw	Transport layer completed before the end of the network layer.
27	NetOpt	Network header options were detected.
26	OffsetDet	A nonzero offset value was detected for the network layer.
25	FragDet	Transport fragmentation was detected at the network layer.
24	NetOflw	Network layer completed before the end of the frame.
23	Attn	Attention Bit: Indicates that one of the following conditions occurred: !MacAddrDet or IpMcst or MacMcst or !Tcplp or !TcpVer4 or 802.3Uflw or RcvEarly or BufOflw or InvalidPreamble or FcsError or DribbleNibble or CodeViolation or TprtChkErr or TprtHdrLenErr or NetHdrChkErr or NetHdrLenErr or NetUflw.
22	IpBcst	The RcvSeq detected an IP broadcast address.
21	IpMcst	The RcvSeq detected and IP multicast address.
20	PauseDet	The received control frame contained a pause command.
19	CtrlFrame	A control frame was received at the special multicast address.
18	MagicDet	A magic wake up frame was detected.
17	MacBcst	The Mac detected a broadcast destination address.
16	MacMcst	The Mac detected a multicast destination address.
15	MacBDet	Frame's destination address matched the contents of MacAddrB.
14	MacADet	Frame's destination address matched the contents of MacAddrA.
13:12	MacId	Id number of the Mac via which this packet was received.
11:09	SessType	The session layer detected by the RcvSeq. 0 - Session is unknown. 1 - Session is NFS/RPC. 2 - Session is FTP-Data. 3 - Session is WWW-HTTP. 4 - Session is NetBios. 5 - Session is reserved. 6 - Session is reserved. 7 - Session is other protocol.
08:06	TprtType	The transport layer detected by the RcvSeq. 0 - Transport is unknown. 1 - Transport is Tcplp or Nlsplpx 2 - Transport is UdpIp or Riplpx. 3 - Transport is NetBiosIp. 4 - Transport is Ncplpx. 5 - Transport is SpxIp. 6 - Transport is SapIp. 7 - Transport is other.

FRAME STATUS A

FIG. 18A

BIT	NAME	DESCRIPTION
05:04	NetVer	<p>The network layer version detected by the RcvSeq.</p> <p>0 - Network version is unknown. 1 - Network version is other. 2 - Network version is 4. 3 - Network version is 6.</p>
03:00	NetType	<p>The combined network and frame layer types detected by the RcvSeq.</p> <p>0 - Frame type is unknown. 1 - reserved. 2 - Frame is 802.3 type. 3 - Unused code. 4 - Frame is 802.3 non-snap. 5 - Frame is 802.3 with Snap header. 6 - Frame is unrecognized ethernet type. 7 - Frame is unrecognized 802.3-snap type. 8 - Frame is ethernet control type with type field = 0x8808. 9 - Frame is 802.3-Snap control type with type field = 0x8808. A - Frame is IPX1 on ethernet type. B - Frame is IPX1 on 802.3-snap type. C - Frame is IPX2 on ethernet type. D - Frame is IPX2 on 802.3-snap type. E - Frame is IP on ethernet type. F - Frame is IP on 802.3-Snap type.</p>

FRAME STATUS A (CONTINUED)

FIG. 18B

KEY TO FIG. 18

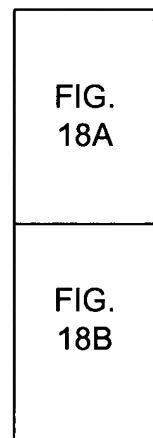


FIG. 18

BIT	NAME	DESCRIPTION
31	802.3Uflw	The frame ended before the 802.3 size/count exhausted.
30	RcvEarly	Data was lost due to insufficient dma bandwidth.
29	BufOflw	The frame length exceeded the capacity of the current buffer.
28	PktMissed	A frame was missed prior to receiving the current frame.
27	CarrierEvent	Refer to E110 Technical Manual.
26	GoodPacket	Refer to E110 Technical Manual.
25	LongEvent	Refer to E110 Technical Manual.
24	InvldPrmbl	Refer to E110 Technical Manual.
23	CrcErr	Refer to E110 Technical Manual.
22	DrblNbb!	Refer to E110 Technical Manual.
21	CodeErr	Refer to E110 Technical Manual.
20	TprtChkErr	A transport layer checksum error was detected.
19	TprtHdrLen	Transport header length error was detected.
18	NetChkErr	A network header checksum error was detected.
17	NetUflw	The frame ended before the Network length was satisfied.
16	NetHdrLen	Network header length error was detected.
15:08	MacHsh	The cumulative XOR of all bytes of the dest mac address of the packet received.
07:00	CtxHsh	The 8-bit context hash generated by exclusive-oring all bytes of the IP source address, IP destination address, transport source port, and the transport destination port.

FRAME STATUS B

FIG. 19

```

#ifndef (C_code)
/*
*****
* The constituents of a TCB block are set forth below. The TCB block
* is shared between the ATCP driver on the host and the NID. The
* TCB block is one contiguous block (180 bytes) of fields in the host
* that gets DMA'd back and forth between host and NID. ULONG is
* four bytes, UCHAR is one byte, and USHORT is two bytes.
*****
*/
struct tcpcb /* { */

    USHORT ip_ckbase;           /* IP base checksum */
    USHORT tcp_ckbase;          /* TCP base checksum of template hdr
                                excluding tcp_seq, tcp_ack & tcp_win
                                and assuming tcp_hl_flg = ACK,
                                and including TCP pseudo-hdr with
                                payload of 20 (std TCP hdrlen)
    */

    ULONG hosttcbaddr;          /* This TCB's address in host mem */
    ULONG hosttcbaddrh;

/*
* The following fields are ordered specifically to match sizes and to match
* the order in which they are read/written.
*/
    ULONG max_rcvwnd;           /* rcv win established by host (sb_hiwat) */
    ULONG max_sndwnd;           /* largest win peer has offered */
    USHORT t_rttmin;             /* minimum rtt allowed */
    USHORT pst_timer;            /* timer count for current PST */
    USHORT t_maxseg;             /* maximum segment size */
    UCHAR t_dupacks;             /* consecutive dup acks recd */
    UCHAR t_shflags;             /* flags shared between BSD & NID */
    ULONG t_rtseq;               /* sequence number being timed */
    ULONG snd_nxt;                /* send next */
    ULONG snd_max;                /* highest sequence number sent; used to recognize retransmits */

    ULONG rcv_adv;                /* advertised window */
    ULONG snd_cwnd;                /* congestion-controlled win */
    USHORT rtr_timer;              /* timer count for current RTR */
    UCHAR t_rxshift;              /* log(2) of persist exp. backoff */
    UCHAR rcv_scale;              /* window scaling for recv window */
    USHORT t_rtt;                  /* round trip time (bumped per tick) */
    USHORT t_srtt;                  /* smoothed round-trip time */
    ULONG snd_una;                  /* send unacknowledged */
    ULONG rcv_nxt;                  /* receive next */

```

```

ULONG rcv_wnd;           /* receive window */
ULONG snd_wnd;           /* send window */
USHORT t_rttvar;          /* variance in round-trip time */
USHORT t_idle;            /* inactivity time (hw?) */
USHORT t_rxtcur;          /* current retransmit value */
UCHAR t_rttupdated;       /* number of times rtt sampled */
UCHAR snd_scale;          /* window scaling for send window */
ULONG snd_wl1;            /* window update seg seq nbr */
ULONG snd_wl2;            /* window update seg ack nbr */
ULONG ts_recent_age;      /* when TS echo last updated */
ULONG ts_recent;          /* timestamp echo data */
ULONG last_ack_sent;      /* rcv_nxt of last Ack */
} tcp_stvars;              /* 100 bytes */

/*
 * Header Template
 */
struct xmit_buffer /* { */
    ULONG reserved1;
    USHORT byte_count;           /* byte count of frame to be xmitted */
    USHORT reserved2;
    ULONG reserved3;
    ULONG link;                 /* link descriptor to next frame */
    struct inic_frame_hdr /* { */
        USHORT tmpht_len;          /* len of template hdr (incl this) */
        UCHAR dhost[6];           /* MAC packet starts here */
        UCHAR shost[6];
        USHORT type;
        UCHAR ip_vhl;
        UCHAR ip_tos;
        USHORT ip_len;
        USHORT ip_id;
        USHORT ip_fragoff;
        UCHAR ip_ttl;
        UCHAR ip_prcl;
        USHORT ip_csum;
        ULONG srcaddr;
        ULONG dstaddr;
        USHORT srcport;
        USHORT dstport;
        ULONG tcp_seq;
        ULONG tcp_ack;
        USHORT tcp_hl_flg;
        USHORT tcp_win;
        USHORT tcp_csum;
        USHORT tcp_urg;
        ULONG tcp_tsopt;           /* timestamp option */
        ULONG pad;                 /* ensures space for VLAN & TS opt */
    } inic_frame_hdr;
} xmit_buffer;              /* 16 + 64 bytes */
#endif /* C_code */

```

KEY TO FIG. 20

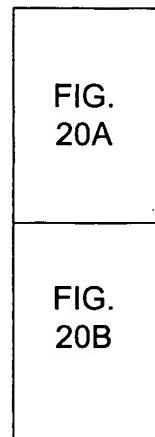


FIG. 20

FIG. 20B